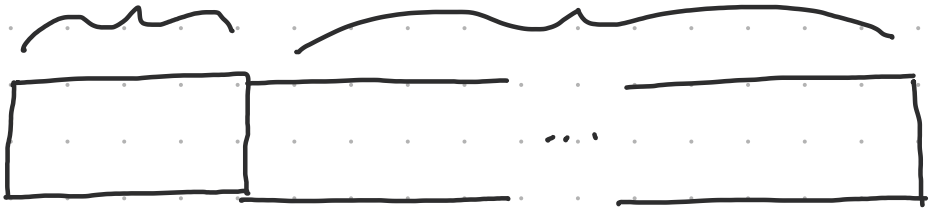
8 byte header n-byte payload



Joe's Lecture malloc™

header = n+1 : n-byte busy block
n+0 : n-byte free block
(n always a multiple of 8)

~ (tilde) is bitwise not

Q1: block_size(17) = 16

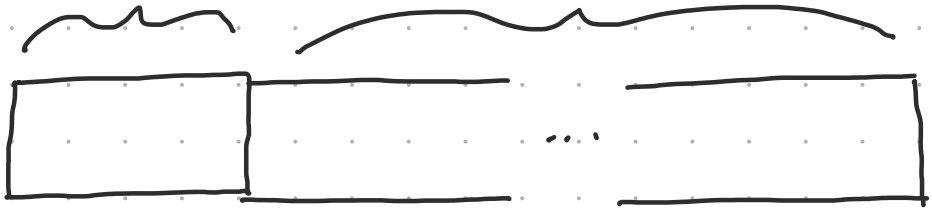
block_busy(17) = true / 1

Q2: block_size(104) = 104

block_busy(104) = false / 0

Q3: Draw the heap after b = malloc(100) in main

8 byte header n-byte payload



Joe's Lecture malloc™

header = n+1 : n-byte busy block
n+0 : n-byte free block
(n always a multiple of 8)

~ tilde
bitwise not

Q1: block_size(17) = 16 (code on handout!)

block_busy(17) = 1 true

Q2: block_size(104) = 104

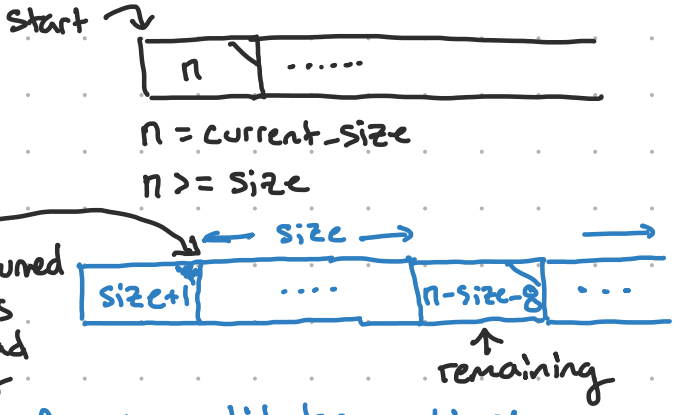
block_busy(104) = 0 false

Q3: Draw the heap after b = malloc(100) in main

```
#define HEAP_SIZE 4096
#define SLOT_SIZE 8
#define HEAP_SLOTS HEAP_SIZE/SLOT_SIZE

uint64_t* HEAP_START = NULL;

void init_heap() { ... } // set up HEAP_START

size_t block_size(uint64_t s) { return s & (~1); }
int block_busy(uint64_t s) { return s & 1; }
size_t round_size(size_t size) { return (size + 7) & ~7; }
```

// Assumes: start is address of the header of a free block of at least size bytes

```
void* allocate_at(uint64_t* start, size_t size) {
  size_t current_size = block_size(start[0]);
  if(current_size > size) {
    uint64_t remaining = current_size - size - SLOT_SIZE;
    int next_block_index = (size / SLOT_SIZE) + 1;
    start[next_block_index] = remaining; // even, free
  }
  start[0] = size | 1; // busy
  return &start[1];
}
```



describe this return value!

```
void* malloc(size_t requested_size) {
  init_heap();
  int val_index = 0;
  size_t rounded = round_size(requested_size);
  while(val_index < HEAP_SLOTS) {
    uint64_t curr_slot = HEAP_START[val_index];
    int current_size = block_size(curr_slot);
    int current_busy = block_busy(curr_slot);
    if(!current_busy && (current_size >= rounded)) {
      return allocate_at(&HEAP_START[val_index], rounded);
    }
    else {
      val_index += (current_size / SLOT_SIZE) + 1;
      continue;
    }
  }
  return NULL;
}
```

makes sure HEAP_START refers to valid heap address

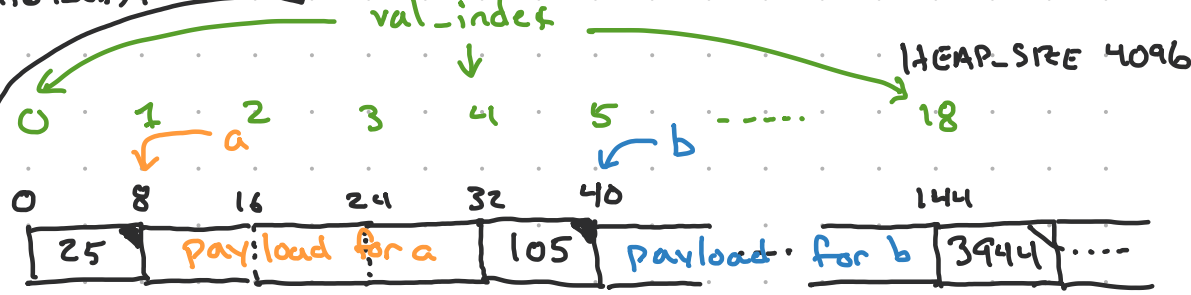
make sure we work in 8-byte payloads

curr_slot:	25	105	3944
current_size:	24	104	3944
current_busy:	true	true	false

↳ called on 3rd iteration (val_index = 18)

+= 4 += 14

```
void free(void* ptr) {
  uint64_t* p = (uint64_t*) ptr;
  p[-1] -= 1;
}
```



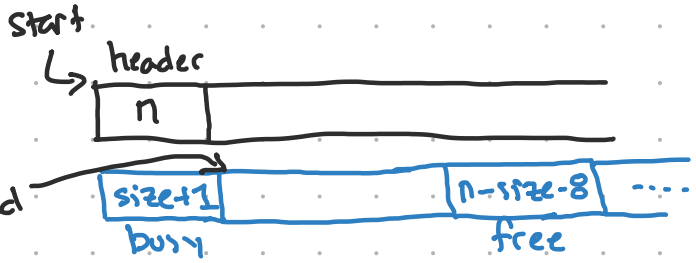

```
int main() {
  int* a = malloc(20);
  int* b = malloc(100);
  int* c = malloc(20);

  free(b);

  int* d = malloc(15);
}
```

```
#define HEAP_SIZE 4096
#define SLOT_SIZE 8
#define HEAP_SLOTS HEAP_SIZE/SLOT_SIZE

uint64_t* HEAP_START = NULL;

void init_heap() { ... } // set up HEAP_START

size_t block_size(uint64_t s) { return s & (~1); }
int block_busy(uint64_t s) { return s & 1; }
size_t round_size(size_t size) { return (size + 7) & ~7; }
```

// ASSUMES start points to free block at least size bytes long

```
void* allocate_at(uint64_t* start, size_t size) {
  size_t current_size = block_size(start[0]);
  if(current_size > size) {
    uint64_t remaining = current_size - size - SLOT_SIZE;
    int next_block_index = (size / SLOT_SIZE) + 1;
    start[next_block_index] = remaining; // even, free
  }
  start[0] = size | 1; // busy
  return &start[1];
}
```

describe this return value! returned (payload address)



size <= n

n = current_size

```
void* malloc(size_t requested_size) {
  init_heap();
  int val_index = 0;
  size_t rounded = round_size(requested_size);
  while(val_index < HEAP_SLOTS) {
    uint64_t curr_slot = HEAP_START[val_index];
    int current_size = block_size(curr_slot);
    int current_busy = block_busy(curr_slot);
    if(!current_busy && (current_size >= rounded)) {
      return allocate_at(&HEAP_START[val_index], rounded);
    }
    else {
      val_index += (current_size / SLOT_SIZE) + 1;
      continue;
    }
  }
  return NULL;
}
```

ensures HEAP_START is set up

rounded%8 == 0

loop over all blocks to find one big enough

expectation: curr_slot is a header

Is the current block
- free?
- big enough?

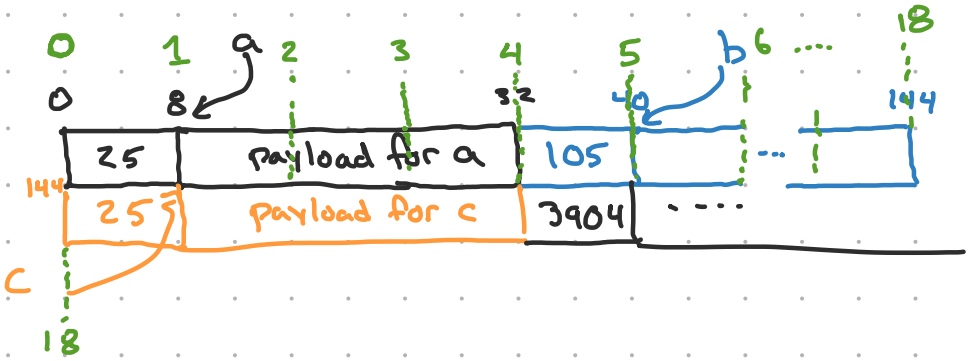
val_index = 0 4 18

current_size = 24 ~~104~~ 3944

└ the free space before we allocate for c

allocate_at called when val_index = 18, current_size = 3944 — what args?

```
void free(void* ptr) {
    uint64_t* p = (uint64_t*) ptr;
    p[-1] = p[-1] - 1;
}
```




```
int main() {
  int* a = malloc(20);
  int* b = malloc(100);
  int* c = malloc(20);
  free(b);
  int* d = malloc(15);
}
```